Federated Learning & Data Privacy, 2024-2025 Second Lab - 4 February 2025

Welcome to the second lab session of the Federated Learning & Data Privacy course! In our first lab, we implemented the Federated Averaging (FedAvg) algorithm, writing the client and aggregator classes, and we performed some preliminary experiments.

I. RECAP OF EXERCISE 3 - The Effect of Local Epochs

Objective: Analyze how the number of local epochs affects the model's performance in a federated learning

setting. **Experiment**:

We ran FedAvg for different numbers of local epochs (e.g., 1, 5, 10, 50, 100). We recorded the test accuracy for each setting.

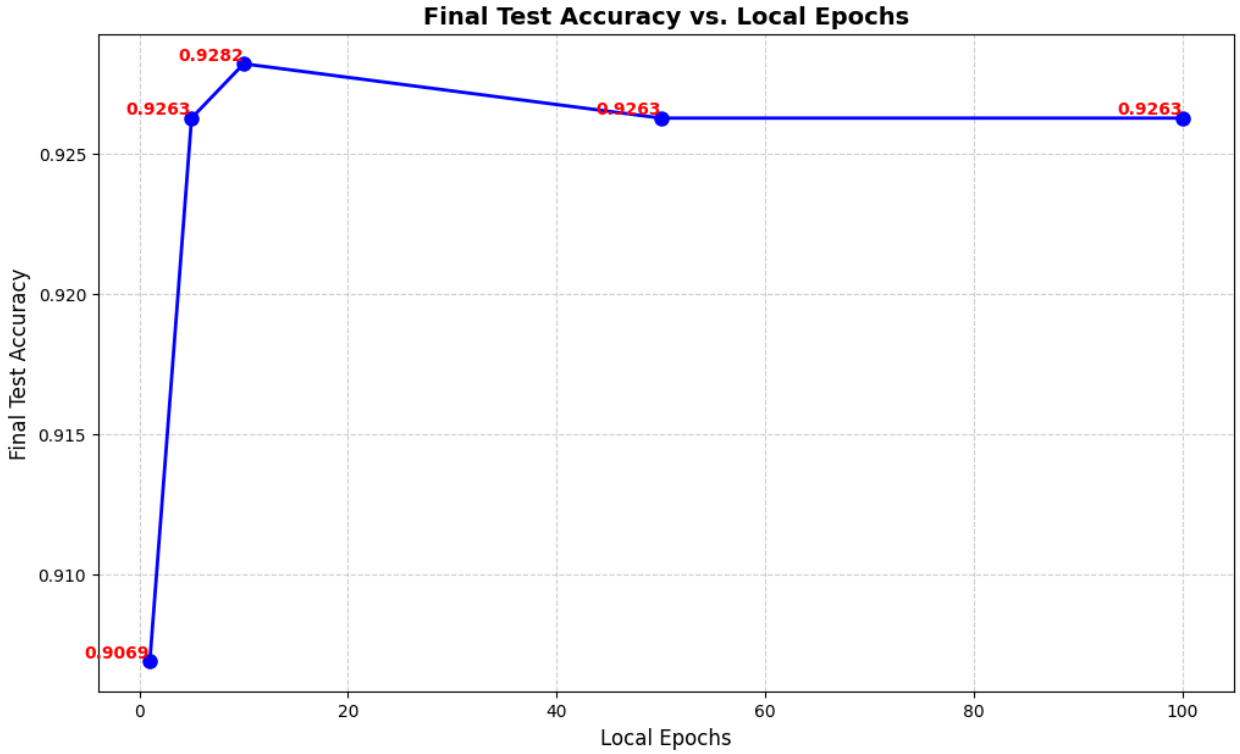
To Save time, I run the code with 20 rounds.

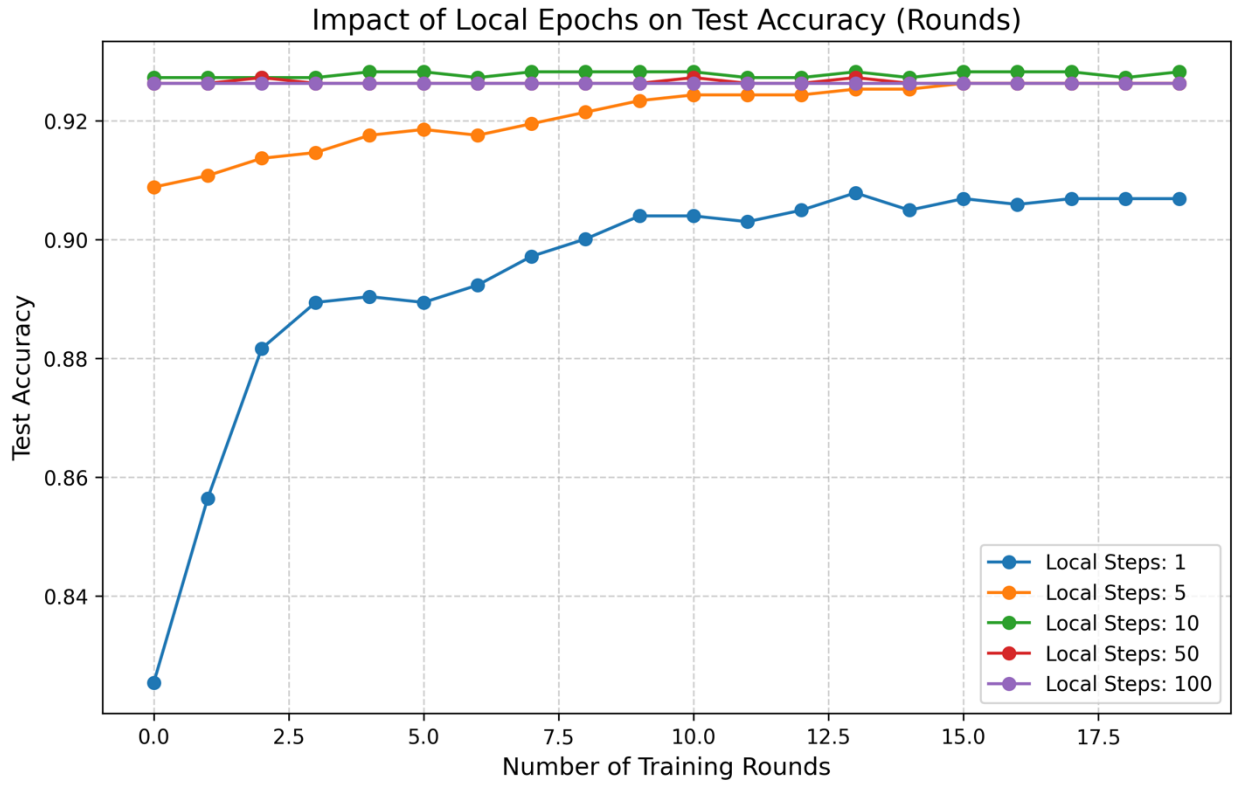
```
echo "=> Train.."

python3 "train.py" \
  --experiment "mnist" \
  --n_rounds 20 \
  --local_steps 1 \
  --local_optimizer sgd \
  --local_lr 0.001 \
  --server_optimizer sgd \
  --server_lr 0.1 \
  --bz 128 \
  --device "cpu" \
  --log_freq 1 \
  --verbose 1 \
  --logs_dir "logs/mnist/" \
  --seed 12 \
  --aggregator_type "centralized"
```

1. Plot:

- We plotted the local epochs on the x-axis and test accuracy on the y-axis.


Final Test Accuracy vs. Local Epochs
0.9282
0.9263
0.9263
0.9263
0.9069
0.925
0.920
0.915
0.910
Final Test Accuracy
0
20
40
60
80
100
Local Epochs


Impact of Local Epochs on Test Accuracy (Rounds)
0.92
0.90
0.88
0.86
0.84
Test Accuracy
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Number of Training Rounds
Local Steps: 1
Local Steps: 5
Local Steps: 10
Local Steps: 50
Local Steps: 100

2. Analysis:

1. Effect of Local Epochs on Model Accuracy

- The results from result.txt show that as the number of local epochs increases, the model's accuracy improves initially and then stabilizes.
- For instance:
  - With Local Steps: 1, the test accuracy steadily increases across rounds, reaching a peak of around 90%.
  - As the local steps increase (e.g., Local Steps: 50), the model achieves higher accuracy faster, stabilizing earlier but not showing significant improvement after a point (around 92%-93%).
- This indicates that increasing the number of local epochs can lead to faster convergence initially, but excessive local training might not yield proportionate gains after stabilization.

2. Expectations

- This result aligns with the expectation that more local training steps reduce the communication overhead in federated learning. However, as more local training is performed, the risk of divergence or overfitting to local client data increases. Therefore, the diminishing returns observed after stabilization are consistent with theoretical predictions.

3. Data Generation and Partitioning in TP1

- **Generation:**
  - Data is sourced from the MNIST dataset using the get_dataset function. This function downloads and combines the training and test datasets into a unified dataset.
  - The combined dataset is then shuffled and a fraction (frac) of it is selected for use.
- **Partitioning:**
  - The iid_split function ensures the data is divided equally among n_clients in an IID (Independent and Identically Distributed) manner. The steps include:
    1. **Random Sampling:** Shuffling the dataset and dividing it into n_clients groups using iid_divide.
    2. **Train-Test Split:** Each client's data is further split into training and testing samples based on the indices.
    3. **Saving:** The resulting data for each client is saved as .npy files using the save_data function.
- **Justification from Code:**
  - In generate_data.py, the argument --iid ensures data is split IID.
  - The iid_split function in utils.py confirms that each client gets approximately equal-sized data, randomly sampled to maintain IID properties.

NEW EXERCISES FOR TP2

The excersies for 4.1 and 4.2 you told us have the problem so I try to finish the excersies 5.1 and 5.2

EXERCISE 5 - Client Sampling
Objective: Implement two client sampling strategies from the research paper "On the Convergence of

FedAvg on Non-IID Data" (https://arxiv.org/abs/1907.02189).

EXERCISE 5.1 - Uniform Sampling Without Replacement Background

Understand uniform sampling as described in Assumption 6. This involves selecting a subset of clients $|S_t| = K$ at each round without replacement. Understand the aggregation formula given by $w_t \leftarrow \frac{N}{K} \sum_{k \in S_t} p_k w^k_t$.

Instructions

1. In aggregator.py, complete the sample_clients() method to uniformly sample self.n_clients_per_round clients from the total available clients.
2. Use self.rng.sample to sample self.n_clients_per_round unique ids from a population ranging from 0 to self.n_clients - 1.
3. Assign the list of sampled ids to self.sampled_clients_ids.
4. Modify the mix() method to:

   Use only the sampled clients for training. For local training, loop over self.sampled_clients_ids instead of all clients.
   Aggregate updates from the sampled clients. Adjust weights accordingly.

Run the code

Run the train.py script with sampling_rate = 0.2.

CODE:

Train.py.

```
def sample_clients(self):
    ...
    # TODO: Exercise 5.1
    self.sampled_clients_ids = self.rng.sample(range(self.n_clients), self.n_clients_per_round)
    self.sampled_clients = [self.clients[id_] for id_ in self.sampled_clients_ids]
```

run.sh

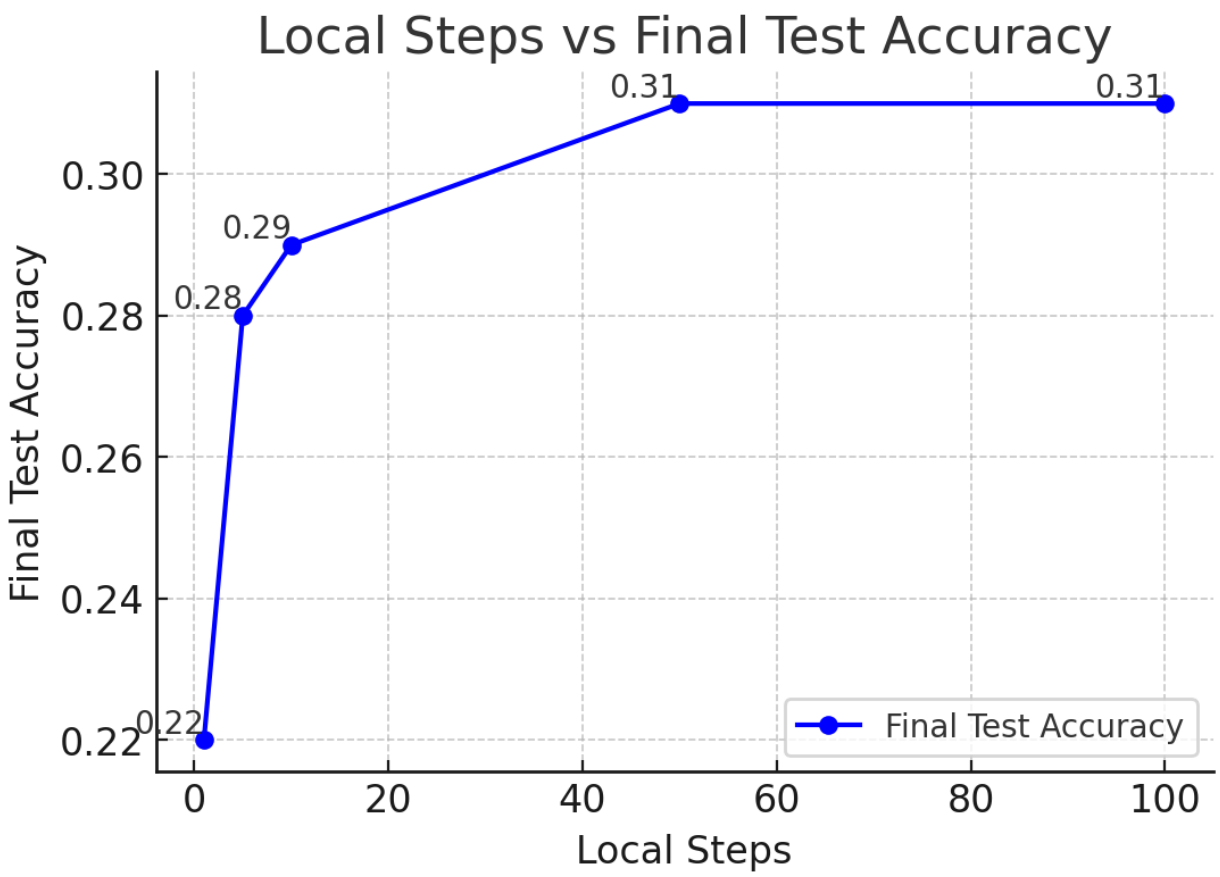
```
    python train.py \
      --experiment "mnist" \
      --n_rounds 5 \
```

```
--local_steps ${local_steps} \
--local_optimizer sgd \
--local_lr 0.001 \
--server_optimizer sgd \
--server_lr 0.1 \
--bz 128 \
--device "cpu" \
--log_freq 1 \
--verbose 1 \
--logs_dir "logs/mnist_local_steps_${local_steps}/" \
--seed 12 \
--sampling_rate 0.2
```

RESULT:

I ran from 1 5 10 50 100 steps locally with 5 laps and it took more than 8 hours to run, here are the results

```
=> Training with 100 local steps...
==> Initialize Clients..
==> Initialize Aggregator..
==> Training Loop..
++++++++++++++++++++++++++++++++++++++++++++++++++
Global | Round 0..
Train Loss: 2.375 | Train Metric: 0.114 |Test Loss: 2.375 | Test Acc: 0.114 |
++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++
Global | Round 1..
Train Loss: 2.114 | Train Metric: 0.311 |Test Loss: 2.114 | Test Acc: 0.311 |
++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++
Global | Round 2..
Train Loss: 2.073 | Train Metric: 0.430 |Test Loss: 2.073 | Test Acc: 0.430 |
++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++
Global | Round 3..
Train Loss: 2.137 | Train Metric: 0.393 |Test Loss: 2.137 | Test Acc: 0.393 |
++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++
Global | Round 4..
Train Loss: 2.125 | Train Metric: 0.317 |Test Loss: 2.125 | Test Acc: 0.317 |
++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++
Global | Round 5..
Train Loss: 2.116 | Train Metric: 0.306 |Test Loss: 2.116 | Test Acc: 0.306 |
++++++++++++++++++++++++++++++++++++++++++++++++++
100%|██████████████████████████████████████████████████| 5/5 [5:44:16<00:00, 4131.39s/it]
```



Final Test Accuracies:

1	0.22
5	0.28
10	0.29
50	0.31
100	0.31

EXERCISE 5.2 - Sampling With Replacement Background

Understand sampling with replacement according to sampling probabilities $p_1, \dots, p_N$. The aggregation formula adjusts to $w_t \leftarrow \frac{1}{K} \sum_{k \in S_t} w^k_t$.

Instructions

1. Extend the sample_clients() method to support sampling with replacement based on self.sample_with_replacement flag.

2. If self.sample_with_replacement is True, use self.rng.choices to sample clients based on their weights self.clients_weights.

Run the code

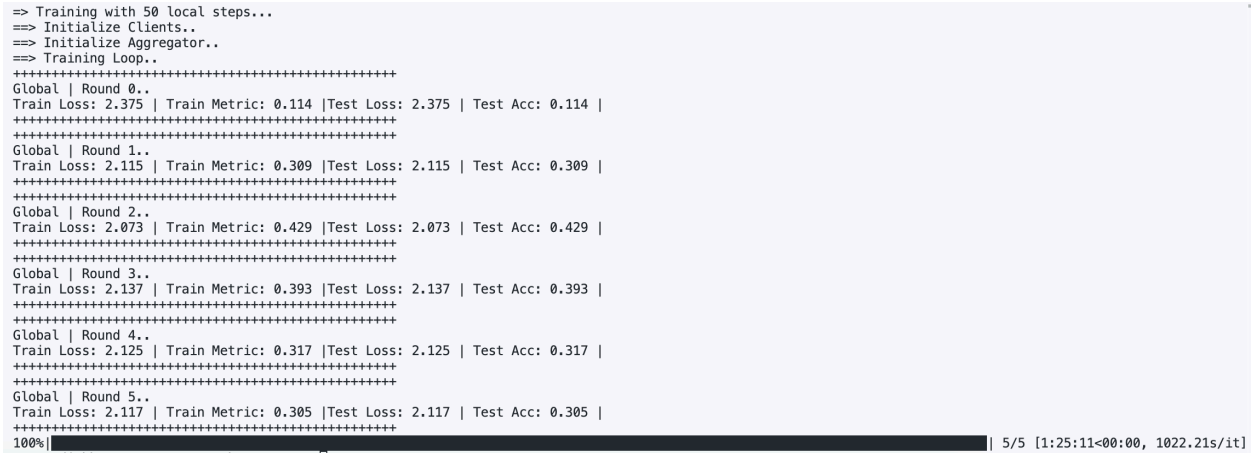
Run the train.py script with sampling_rate = 0.2 and sample_with_replacement = True. Good luck, and don't hesitate to ask questions and collaborate with your peers!

Code:

```
if self.sample_with_replacement:
    # Sample with replacement using weights
    self.sampled_clients_ids = self.rng.choices(
        population=range(self.n_clients),
        weights=self.clients_weights,
        k=self.n_clients_per_round
    )
else:
    # Sample without replacement
    self.sampled_clients_ids = self.rng.sample(range(self.n_clients), self.n_clients_per_round)

self.sampled_clients = [self.clients[id_] for id_ in self.sampled_clients_ids]
```

RESULT



Final Test Accuracies:

1	0.22
5	0.28
10	0.29
50	0.31

I ran the code of 5.1 and my laptop too hot so I just run the code of 5.2 with 50 local steps.